Introdução à Teoria dos Grafos

Algoritmos de Busca

Prof. Pablo De Chiaro
algoritmos e estrutura de dados 2 (AEDII)

_aula 6

Grafos

```python
# ----------------------------
# Grafo (não-direcionado) com custos (km)
# ----------------------------
GRAPH = {
    "Pelotas": {"Camaquã": 150, "Rio Grande": 55, "Bagé": 180, "Santa Maria": 370},
    "Camaquã": {"Pelotas": 150, "Guaíba": 65, "Porto Alegre": 125},
    "Guaíba": {"Camaquã": 65, "Porto Alegre": 30},
    "Porto Alegre": {"Guaíba": 30, "Camaquã": 125, "Rio Grande": 320, "Bagé": 380, "Santa Maria": 290},
    "Rio Grande": {"Pelotas": 55, "Porto Alegre": 320},
    "Bagé": {"Pelotas": 180, "Porto Alegre": 380},
    "Santa Maria": {"Pelotas": 370, "Porto Alegre": 290},
}
```

Heurística

```python
# ----------------------------
# Coordenadas aproximadas (lat, lon) p/ heurística (linha reta)
# fórmula de Haversine - distância em linha reta entre duas cidades na superfície da Terra
# ----------------------------
COORDS = {
    "Pelotas": (-31.7710, -52.3420),
    "Camaquã": (-30.8489, -51.8051),
    "Guaíba": (-30.1086, -51.3233),
    "Porto Alegre": (-30.0346, -51.2177),
    "Rio Grande": (-32.0349, -52.1071),
    "Bagé": (-31.3297, -54.1063),
    "Santa Maria": (-29.6868, -53.8149),
}
```

Heurística: usamos em algoritmos, aqui para estimar o quão perto um estado (ex.: cidade, nó) está do objetivo. **Não é o custo real, é só uma aproximação.** Serve para guiar a busca, tornando-a mais rápida ao priorizar caminhos promissores.

Algoritmo da heurística de Haversine

```python
# fórmula de Haversine - distância em linha reta entre duas cidades na superfície da Terra
def haversine_km(a, b):
    lat1, lon1 = COORDS[a]
    lat2, lon2 = COORDS[b]
    R = 6371.0
    p1, p2 = math.radians(lat1), math.radians(lat2)
    dphi = math.radians(lat2 - lat1)
    dlmb = math.radians(lon2 - lon1)
    x = math.sin(dphi / 2) ** 2 + math.cos(p1) * math.cos(p2) * math.sin(dlmb / 2) ** 2
    return 2 * R * math.asin(math.sqrt(x))
```

Heurística Simplificada

```python
# Heurística simplificada
# Distância aproximada até Porto Alegre (em km)
HEURISTICA_POA = {
    "Porto Alegre": 0,
    "Pelotas": 235,
    "Camaquã": 125,
    "Guaíba": 30,
    "Rio Grande": 320,
    "Bagé": 380,
    "Santa Maria": 290,
}
```

```
h = HEURISTICA_POA[cidade]
```

Algoritmos

Greedy Best-First (Busca Gulosa)

Como decide: sempre vai para a cidade que parece mais próxima do destino (usa a heurística).

Ponto forte: é rápida, porque olha só a proximidade.

Limitação: pode escolher um caminho que parece bom, mas sai mais caro no total.

Algoritmos

DFS (Depth-First Search)

Como decide: mergulha em um caminho até o fim antes de voltar atrás.

Ponto forte: simples e pode usar menos memória em alguns casos.

Limitação: não garante o caminho mais curto nem o mais barato.

Algoritmos

Dijkstra

Como decide: sempre escolhe expandir o caminho de menor custo acumulado.

Ponto forte: garante o caminho mais barato (ótimo) se todos os custos forem positivos.

Atividade em Individual ou Duplas

- Construa o grafo das cidades apresentado em aula usando a ferramenta https://graphonline.top/pt/create_graph_by_edge_list. Insira os vértices (cidades) e as arestas (estradas) com os custos aproximados em quilômetros.

- Implemente, em Python, os algoritmos BFS e DFS para encontrar um caminho entre Pelotas (início) e Porto Alegre (objetivo). Registre o caminho encontrado e o número de passos percorridos em cada algoritmo.

- Implemente, em Python, o algoritmo de Dijkstra considerando os custos das arestas. Compare o caminho obtido e o custo total (em km) com os resultados do BFS e DFS.

Atividade em Individual ou Duplas

Entrega 16/09 (sem prazo extra)
Via Blackboard, todos entregam!

obrigatório envio do código + print do Grafo visual
não compactar (.zip, .rar)

Atenção: **Você poderá ser chamado para explicar o algoritmo que escreveu.**

Introdução à Teoria dos Grafos

parte 2

Prof. Pablo De Chiaro
algoritmos e estrutura de dados 2 (AEDII)

_aula 6